Package, javadoc, introduzione alle API

Stefano Mizzaro

Dipartimento di matematica e informatica
Università di Udine
http://www.dimi.uniud.it/~mizzaro
mizzaro@dimi.uniud.it

5 maggio 2003

Riassunto: OO in Java

- `class`
- `public`, `private`, `protected`
- costruttore, `new`
- `this`
- *dot notation* (`C.m()`, `x.m()`)
- `extends`
- `super`
- `instanceof`
- Maniglie
- `abstract`
- `interface`
- `implements`

Scaletta

- ***Package***
- Javadoc
- API
  - Package principali
  - Classi principali
  - Documentazione con javadoc

Package

- Insieme di classi e interfacce
- "Namespace"
- Struttura gerarchica
- Basata sul file system
- Nomi univoci grazie al dominio internet
- Due istruzioni:
  `package` e `import`

`package`

```
package <nomepackage>;
```

- Prima istruzione di un file
- In quale package mettere le classi (e le interfacce) del file
- Se l'istruzione non c'è ⇒ package "di default"

`import`

- Dice in quali package/classe cercare i nomi usati nelle classi di questo file

```
import <nomepackage>.<nomeclasse>;
import <nomepackage>.*;
```

- Esempi
  - `import java.util.Calendar;`
  - `import java.util.*;`
  - `import java.io.*;`
- N.B.: `import java.*;`

Full qualified name

- `<nomepackage>.<nomeclasse>`
- `java.util.Calendar`
- `java.applet.Applet`
- Quindi: l'`import` serve per evitare di scrivere i full qualified name ogni volta

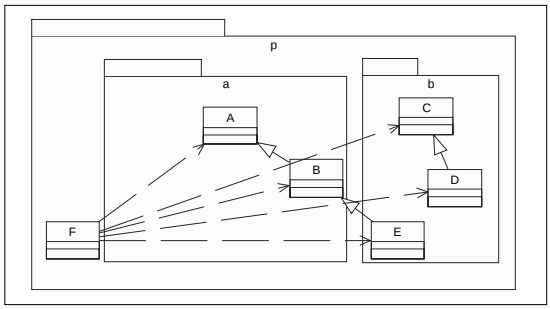
Package e file system

- La struttura gerarchica dei package corrisponde a quella del file system
  - Le classi del package `a` vanno nella directory `a`
  - Le classi del package `a.b` vanno nella directory `a/b`
  - …
  - I file del package `xxx.yyy.zzz` vanno nella directory `xxx/yyy/zzz`
  - E la radice deve essere in `CLASSPATH`

Esempio

- Nel package `p.a` (directory `./p/a`) ci sono 2 classi `A` e `B` (sottoclasse di `A`)
- Nel package `p.b` (directory `./p/b`) ci sono 3 classi `C`, `D` (sottoclasse di `C`) ed `E` (sottoclasse di `B`)
- Nel package `p` (directory `./p`) c'è la classe `F` (che usa tutte le altre classi)

Diagramma dei package



Package, compilazione, esecuzione

- Bisogna ***risalire alla radice***
- Per compilare (usare nomi di file)

```
>javac -d bin -verbose p/F.java
```

```
>cd p; javac F.java
```
(non trova i package)

- Per eseguire (usare *full qualified names*)

```
>java p.F
>cd p; java F
```
(non trova `F`, vuole `p.F`)

```
>cd p; java p.F
```
(non va, cerca il package/dir `p`)

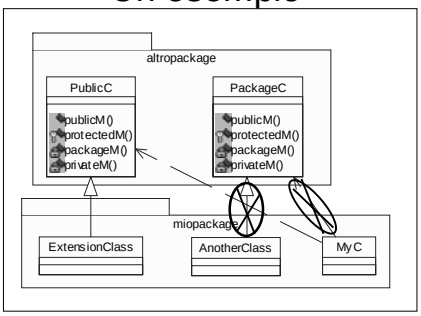
Nomi univoci di package

- Nome di package: `xxx.yyy.zzz` (minuscole!)
- Nomi che iniziano con "`java.`" sono riservati (`java.lang`, `java.io`, `java.util`, …)
- Dal nome del dominio internet: il nome di un package sviluppato in `uniud.it` inizia con `it.uniud`
- e prosegue con un nome esplicativo deciso dal programmatore: `it.uniud.twm`, `it.uniud.twm.progetto`, …

Package e visibilità

- Attributi e metodi possono avere visibilità:
  - `public`: ovunque, anche in altri package
  - `protected`: solo nelle sottoclassi **e nel package**
  - (niente) "package": solo nel package
  - `private`: solo nella classe
- Classi e interfacce possono avere visibilità:
  - `public`: anche da altri package
  - (niente) "package": solo nel package

Un esempio



I modificatori (non tutti!)

	Metodo	Attributo	Classe	Interfaccia
`public`	●	●	●	●
`private`	●	●		
`protected`	●	●		
(niente)	●	●	●	●
`abstract`	●		●	●
`final`	●	●	●	
`static`	●	●		

Scaletta

- Package
- ***Javadoc***
- API

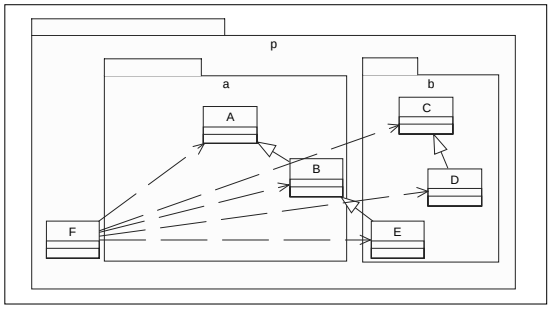
Javadoc

- Generazione automatica della documentazione
- A partire dai commenti `/** ... */`

  `javadoc [<nomefile> | <nomepackage>]*`

  produce la documentazione per `<nomefile>` e/o `<nomepackage>`
- Tipicamente in HTML (configurabile: ***doclet***)

Esempio (di nuovo)



Generazione della documentazione

```
>javadoc -verbose -d doc p p.a p.b
```

- Facciamolo! (…\lucidi\materiali\packages)
- Esaminiamola velocemente…
- Aggiungiamo commenti…

Documentazione e OO

- Documentazione utile anche durante la programmazione
  - Per classi, interfacce, metodi, … definite dal programmatore (trovare le definizioni nella gerarchia)
  - Per le API
- Scrivere (e generare) la documentazione ***insieme al codice!!***
  - Un commento per ogni classe, interfaccia, metodo o attributo `public`

Scaletta

- Package
- Javadoc
- ***API***

Rassegna API?

- **2991** classi in **135** package
- Gulp!
- Diamo “solo” uno sguardo mirato…

Fonti utili

- Documentazione in linea delle API
  - Creata con javadoc
  - In formato HTML
  - Navigabile con un browser
  - (…java/docs/)
- Sorgenti (.java) delle API!
  - File src.zip
  - (…java/src/)

Un po’ di storia…

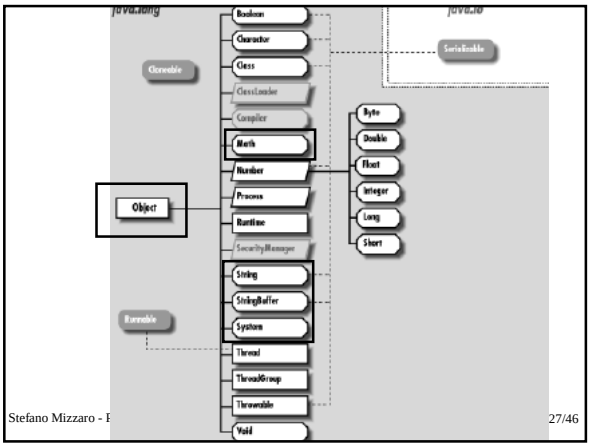
Anno	Vers.	# classi	# package	Altro
1991	Oak			Elettrodomestici, …
1995	α			HotJava, applets
1996	1.0	**212**	**8**	Web browser
1997	1.1	**504**	**23**	
1998	1.2	**1520**	**59**	Java 2: Swing, collections
2000	1.3	**1842**	**76**	Bug fix, HotSpot
2003	1.4	**2991**	**135**	NIO, pattern match & r.e., XML (DOM/SAX), assert, …

Un po' di package (usati spesso)…

- `java.lang`: (importato implicitamente)
- `java.util`: (date, contenitori di oggetti, …)
- `java.io`: input/output, flussi, file, …
- `java.awt`, `java.awt.event`, `javax.swing` (GUI)
- `java.applet` (applet in pagine Web)
- `java.sql` (JDBC x accesso DB)
- `java.rmi` (Remote Method Invocation)
- …

Un po' di `java.lang`…

- `Object`
- `System`
- `Math`
- `String`



`java.lang.Object`

- Radice della gerarchia, superclasse di tutte le classi
- I suoi metodi sono ereditati da tutte le classi
  - `public String toString()`
  - `public boolean equals(Object o)`
  - `protected Object clone()`
  - …

`toString`

- `public String toString()`
- Restituisce una rappresentazione testuale, visualizzabile dell'oggetto
- Chiamato ***implicitamente*** quando serve
- Da sovrascrivere in ogni classe che definiamo
- (vediamo com'è fatto…)

Esempio

```
class Punto {
  // ...
  public String toString () {
    return "Punto: (" + getX() + ", " + getY() + ")";
  }
}
```

```
class ProvaPunto {
  // ...
  public static void main(String[] args) () {
    Punto p = new Punto(2.3, 5.67);
    System.out.print(p);
    System.out.println(p);
  }
}
```

```
>java ProvaPunto
Punto: (2.3, 5.67)…
```

equals

- `public boolean equals(Object o)`
- 2 tipi di uguaglianza fra `x` e `y`:
  - Essere lo stesso oggetto (`x == y`)
  - Essere 2 oggetti uguali (`x.equals(y)`)
- `==` confronta solo i riferimenti!!
- In realtà, l'`equals` di `Object` è come l'`==`…
- …ma `equals` può essere sovrascritto!

Esempio (1/3)

```
class Punto {
  // ...
  public boolean equals (Object o) {
    return (((Punto)o.x) == this.x &&
            ((Punto)o).y == this.y);
  }
}
```

- Notate il downcast: devo sovrascrivere, e il parametro di `equals` è `Object`…
- Si può fare di meglio…

Esempio (2/3)

```
class Punto {
  // ...
  public boolean equals (Object o) {
    if(o instanceof Punto)
      return (((Punto)o).getX() == this.getX() &&
              ((Punto)o).getY() == this.getY());
    else
      return false;
  }
}
```

- Ancora meglio…

Esempio (3/3)

```
class Punto {
  // ...
  public boolean equals (Object o) {
    return (o instanceof Punto &&
            ((Punto)o).getX() == this.getX() &&
            ((Punto)o).getY() == this.getY()
           );
  }
}
```

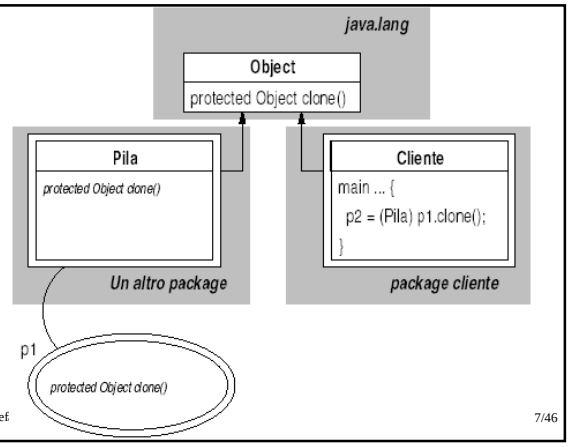

clone

- `protected Object clone()`
- Duplica un oggetto
- Come per l'`equals`, 2 tipi di copiatura:
  - `x = y` copia solo la maniglia (alias)
  - `x = y.clone()` crea una nuova copia
- `clone` può essere sovrascritto in ogni classe che vogliamo rendere clonabile
- Ma è `protected`…

La clonazione: perché l'= non va

- Proviamo a clonare una `Pila`
- `p2 = p1` crea un alias
- Bisogna usare `clone`, che però è `protected`…

```
class Cliente {
  public ... main ...{
    ...
    Pila p1, p2;
    p1 = new Pila();
    ...
    // ... modifiche a p1...
    p2 = p1;
    // ... altro codice...
  }
}
```



clone (2/3)

- Sostituire `p2 = p1` con `p2 = (Pila) p1.clone()`?
  - (il cast perché `clone` restituisce un `Object`)
- Non va:
  - `clone` della `Pila p1` è ereditato da `Object`, in cui è definito `protected`
  - è visibile solo nel package di `Object` o nelle sottoclassi di `Pila`
  - `Cliente` vede il metodo `clone` di `Object`, non di `Pila`!

clone (3/3)

- Soluzione: `Pila` ri-implementa `clone` e lo rende `public`
- Ri-implementazione standard: uso il `clone` della sopraclasse

```
public Object clone() {
        throws CloneNotSupportedException {
  return super.clone();
}
```


Osservazioni:

- L'unica differenza è la visibilità
- Non si può definire un `clone` che restituisce qualcosa di diverso da `Object`
- `clone` di `Object` fa una copia superciale (shallow copy), non profonda (deep).
- Se serve una copia profonda, bisogna modificare (sovrascrivendolo) `clone`

java.lang.System

- Sveliamo un mistero: perché
  - `System.out.println`
  - `System.in.read`
- Classe `System`
- Variabile di classe `out (in)`

```
public final static PrintStream out;
```

- Metodo d'istanza `println (read)`

java.lang.Math

```
public static double sin(double)
public static double cos(double)
public static double sqrt(double)
public static double log(double)
public static double abs(double)
public static double exp(double, double)
public static double floor(double)
public static double ceil(double)
public static double min(double, double)
public static double max(double, double)
...
```

java.lang.String

- Sequenza di caratteri
- `String`: sottoclasse `final` di `Object`
- Letterali stringa: `""` (chiamata implicita del costruttore)
- `+`: concatena stringhe

java.lang.String

- `String`: modificabili
- (e `StringBuffer`: non modificabili)
- `public int length()`
- `public int charAt(int)`
- `public boolean equals(Object)`
- `public String substring(int)`
- `public String substring(int,int)`
- `public char[] toCharArray()`

Riassunto

- Package
- Javadoc
- API
  - `Object`
    - `toString, equals, clone`
  - `System`
  - `Math`
  - `String`

Parole riservate (di nuovo)

```
abstract boolean break byte case
catch char class const continue
default do double else extends false
final finally float for goto if
implements import instanceof int
interface long native new null
package private protected public
return short static super switch
synchronized this throw throws
transient true try void volatile
while
```